Distributed load balancing using Alternating Direction Method of Multipliers

Alireza Sheikhattar and Mehdi Kalantari
Electrical and Computer Engineering Department
University of Maryland, College Park, MD 20742, USA
Email: \{arsha89, mehkalan\}@umd.edu

Abstract—Load balancing is a major issue in networking, where the traffic load is effectively distributed across multiple paths of a network graph. Practical networks have recently grown larger in size and complexity and demand, which necessitates the notion of decentralized load balancing. This paper provides an algorithm for balancing the traffic load in a general network setting. Unlike most of state-of-the-art algorithms in load balancing context, the proposed method is fully distributed. The effective distribution of loads is realized through solving a convex optimization problem where the $p$-norm of network load is minimized subject to network physical constraints. We employ the Alternating Direction Method of Multipliers (ADMM) to solve the $p$-norm problem in a distributed fashion, where each node only exchanges information with its neighboring nodes. Numerical results show that the proposed algorithm requires very few communication steps to balance the traffic load on bottleneck for random network graphs.

I. INTRODUCTION

As communication networks scale up in size, effective distribution of the traffic load throughout the network is a matter of great importance. By spreading the traffic load across the network, load balancing maximizes the network throughput and makes it possible to utilize resources more uniformly. In wired networks, balancing the traffic ensures effective use of links’ capacities, and consequently reduces the likelihood of congestion. In wireless sensor networks, load balancing reduces congestion hot spots as the traffic load is dispersed uniformly across multiple paths, leading to more balanced resource utilization and energy consumption of sensor nodes. This will prolong the energy lifetime of the sensor network, which is a crucial fact, in particular in cases when the devices are battery powered.

Load balancing has been proven as an active research area for decades and received much attention recently by emerging large-scale sensor networks. For example, in [1] load balancing in dense wireless networks has been studied, in which the authors consider a minimax optimization problem and develop lower bounds for the objective of the achievable minimum. In [2], a node-centric load balancing algorithm is designed which constructs a load-balanced tree in sensor networks with an asymmetric architecture. This algorithm distributes the traffic generated by sensor nodes evenly across the different branches of the routing tree. Several algorithms have been proposed to solve load balancing problem [3]–[5]. These methods, however, hold the notion of centralized processing in common. This means that all information is aggregated at a central or fusion node for processing. Data collection and transmission to a central processing node pose a major drain on communication and energy resources. Furthermore, the central node that performs load balancing introduces a single point of failure (SPOF) which reduces the reliability of communication. To overcome these shortcomings, it is desirable to replace the notion of centralized processing with decentralized cooperation of network nodes in a distributed scenario. By \textit{distributed} we imply that a central node no longer exists and there is no global network information available at any location. Furthermore, each node communicates only with its neighbors; therefore, it has one-hop access to the network information.

To the best of our knowledge, there is no load balancing algorithm offered in the distributed context. The main goal of this paper is to devise a fully distributed algorithm to balance the traffic load in a general network setting, in particular on the bottleneck links where congestion is more likely to occur. The balanced flow assignment is achieved through solving a convex optimization problem where \textit{$p$-norm} of network flow is minimized as cost function constrained to flow conservation law. In the \textit{$p$-norm} problem, \textit{"p"} determines the balance level of load distribution. The case $p = 1$ is equivalent to the shortest path routing where no load balancing occurs. By increasing the parameter $p$, traffic patterns exhibit more balanced load distribution. In this case, each node distribute the incoming traffic (including the generated traffic) through multiple paths instead of forwarding to the node in the shortest path direction. The optimization solution will tend to minimax problem solution for the limiting case when $p \rightarrow \infty$. In most practical scenarios, the solution for the \textit{$p$-norm} problem with sufficiently large $p$ will be a good approximation of the maximally balanced flow.

\textbf{Contribution}: Our main objective in this paper is to design a distributed scheme to solve the \textit{$p$-norm} optimization in a network setting. Such distributed scheme follows an iterative procedure where the iterations converge to the optimal solution of the \textit{$p$-norm} problem. Our solution for this problem relies on the Alternating Direction Method of Multipliers (ADMM), which is a powerful algorithm for solving convex optimization problems, see [6] for survey.

ADMM provides a three-step iterative scheme when applied
to the suitably reformulated form of p-norm problem. These three iteration steps are as follows: Distributed projection, parallel flow minimization, and dual update. The distributed implementation of these three steps are discussed thoroughly. We present a distributed procedure for the projection step by reducing it to solving system of linear equations. A significant desired feature pertaining distributed iterative schemes is their fast convergence rate. Hence, an accelerated second-order iterative scheme, presented in our previous work [7], is employed to solve linear subproblems in an efficient distributed way. To perform these iterations, all nodes should have access to a certain parameter which relies on the network topology. To find this parameter, we use a distributed initialization and network identification, which is based on novel idea of distributed estimation of desired parameter by modeling network as a multi-input multi-output (MIMO) dynamical system. By this initialization setup, we claim that each node can make an accurate evaluation of the desired parameter only by exchanging information with its neighboring nodes. Simulation results show that our proposed algorithm requires very few (typically, 3-5) ADMM iterations to achieve a load-balanced bottleneck links, even when used on randomly generated complex network topologies.

The rest of the paper is organized as follows. We will present our network model and assumptions in Section II. In Section III, we formulate load balancing as p-norm optimization problem and present an ADMM-based iterative scheme to solve p-norm problem. The distributed implementation of the proposed algorithm is further given in Section III, where projection subproblems, and initialization stage are discussed. Numerical examples and simulation results are presented in Section IV. Finally, Section V concludes the paper.

II. NETWORK MODEL

We consider a network model based on a connected directed graph \( G = (V, E) \) with \( V = \{1, 2, \ldots, N+1\} \) as the set of vertices and \( E \subset V \times V \) as the set of edges. In this model, network nodes and communication links are represented by vertices and edges, respectively. Each edge is denoted by \((i, j) \in E\), with \( i < j \), means that nodes \( i \) and \( j \) are neighbors which can exchange information directly. The set of one-hop neighbors of node \( i \) is denoted by \( N_i \); the cardinality of this set, \( d_i = |N_i| \), defined as the degree of node \( i \). Assume there is \( M \) links available in the network graph.

Let \( f \) be the flow vector containing network flows \( f_m, m = 1, 2, \ldots, M \) streaming on the links. The sign of flow \( f_m \) corresponding to the link \((i, j)\) determines the direction of flow from node \( i \) to \( j \), with \( f_m < 0 \) meaning that flow is streaming in the opposite direction. Network nodes are fed by external source vector \( b \), where each component \( b_i \) denotes the incoming \( (b_i > 0) \) or outgoing \( (b_i < 0) \) flow rate at the \( i \)-th node. The sign of component \( b_i \) determines the role of node in the network with positive sign as the source node, negative sign as the sink node and zero-valued for intermediate relay nodes. We know that \( \sum_{i \in V} b_i = 0 \). The vector \( b \) is assumed to be a feasible source vector meaning that there exists at least one flow vector \( f \in F(b) \) satisfying channel capacity constraints.

The basic feasibility condition for our network model is the flow conservation law. This implies that the sum of input flows to a node, including the generated flow at that node, is equal to zero. The flow conservation can be written in the compact form, \( Kf = b \) where matrix \( K \) is the incidence matrix of size \((N+1) \times M\) for network graph. Each column of \( K \) is associated to a single edge: the column associated to the link \((i, j) \in E\) has +1 at the \( i \)-th entry and -1 at the \( j \)-th entry; the remaining entries have zeros.

We know that rank of the incidence matrix \( K_{(N+1) \times M} \) for a typical connected graph is equal to \( N \) [8]. In order to obtain an independent set of equations, one node is specified as the reference node and the corresponding equation is eliminated to achieve

\[
Kf = b
\]

where \( K_{N \times M} \) is the reduced incidence matrix of full-row rank. The reduced source vector \( b \) of size \( N \) specifies the generated flow rates at all nodes excluding the reference node.

In this paper, we propose a distributed method based on ADMM iterations to solve p-norm optimization problem, where each node can only exchange information with neighboring nodes.

III. DISTRIBUTED LOAD BALANCING ALGORITHM

In this section, we will present a reformulation of our optimization problem. Among all possible reformulations for ADMM, we choose the one which best match our aim of distributed implementation. We will show how application of ADMM would yield distributed, fast, and numerically stable iterations.

A. ADMM formulation of p-norm Problem

The p-norm problem can be formulated as an equality-constrained convex optimization problem as follows

\[
\begin{align*}
\text{minimize} & \quad J_p(f) = \|f\|_p \\
\text{subject to} & \quad Kf = b
\end{align*}
\]

where we seek the least solution in the \( \ell-p \) sense of the underdetermined linear system \( Kf = b \). The p-norm, \( p \in [1, \infty) \), of vector \( f \in \mathbb{R}^M \) can be defined as \( \|f\|_p = \left[ \sum_{m=1}^{M} (f_m)^p \right]^{1/p} \), where \( f_m \) is the flow component on the \( m \)-th link.

One way to solve p-norm problem is to apply the ADMM algorithm. In order to make the p-norm problem perfectly match the ADMM form [6], we first formulate (2) as follows

\[
\begin{align*}
\text{minimize} & \quad I_C(x) + \|z\|_p^p \\
\text{subject to} & \quad x - z = 0
\end{align*}
\]

where \( I_C(x) \) is the indicator function of constraint set \( C = \{x \in \mathbb{R}^M \mid Kx = b\} \), which is equal to 0 when constraint is satisfied \( x \in C \), and takes value \( +\infty \) if constraint is violated \( x \notin C \). By this formulation, the constraint appears in the objective term as the indicator function on the duplicate variable \( x \) enforcing the optimal solution to satisfy the constraint...
where \( x \in C \), while minimizing the p-norm cost function. For ease of formulation, ADMM algorithm for our reformulated problem (3) can be expressed in scaled form [6] as follows

\[
\begin{align*}
\alpha^{k+1} &= \arg\min_x \left( I_c(x) + \frac{\rho}{2} \|x - z^k + u^k\|^2 \right) \\
z^{k+1} &= \arg\min_z \left( \|z\|^p_p + \frac{\rho}{2} \|x^{k+1} - z + u^k\|^2 \right) \\
\beta^{k+1} &= u^k + x^{k+1} - z^{k+1}
\end{align*}
\]

(4) (5) (6)

where \( u^k = (1/\rho) y^k \) is the scaled dual variable.

The minimization steps (4) and (5) can be efficiently simplified. The minimization on the indicator function (4) takes the form of projection on the constraint set

\[ x^{k+1} = \Pi_c(z^k - u^k) \]

(7)

where \( \Pi_c \) is the projection operator on the constraint set \( C \).

The minimization in z-update step (5) can be carried out in a parallel way to separate scalar minimizations on components \( z_m \). This is due to the fact that the p-norm function has the component separability feature, where the objective function can be separated as summation of functions on individual components, \( \|z\|^p_p = \sum_{m=1}^M (z_m)^p \). This separability property alongside the separable quadratic term in (5) will result in a separable \( I_p(x^{k+1}, z, u^k) \) and the opportunity to update \( z^{k+1} \) in a completely parallel way

\[ z^{k+1} = \sum_{m=1}^M \arg\min_{z_m} \left( (z_m)^p + \frac{\rho}{2} (x^{k+1}_m - z_m + u^k_m)^2 \right) \]

(8)

In order to avoid the ambiguity of multiple solutions for minimization, \( p \) is assumed to be an even number. This scalar minimization can be easily solved at each node for all flow components associated with the connected links. The parallelization of z-update step on network flow components perfectly matches up to the aim of decentralization and makes it suitable for a distributed algorithm design. In the final step of ADMM iteration (6), the dual variable is updated.

In summary, ADMM iterations for our case can be expressed in the following three steps: (1) Projection on the constraint set, (2) Parallel flow minimization, and (3) Dual update

\[
\begin{align*}
x^{k+1} &= \Pi_c(z^k - u^k) \\
z^{k+1} &= \sum_{m=1}^M \arg\min_{z_m} \left( (z_m)^p + (\rho/2)(x^{k+1}_m - z_m + u^k_m)^2 \right) \\
\beta^{k+1} &= u^k + x^{k+1} - z^{k+1}
\end{align*}
\]

(9)

It should be noted that the convergence of ADMM to the optimal solution is independent of initial values of variables; however, a good initial point can significantly reduce the number of iterations required to converge within a desired accuracy. In the proposed ADMM iterations, the two last steps involve straightforward computations such as additions, multiplications, and simple root-finding process, which can be carried out locally at each node without the need of exchanging information; however, the projection step has more complicated computations involving matrix inversion which requires some communication steps among nodes. In the next section, we will show how the projection step needs to be specially designed to perfectly match the distributed setting.

**B. Distributed Projection Subproblems**

The projection onto set \( C = \{x \in \mathbb{R}^M | Kx = b\} \) can be casted as a minimum Euclidean norm problem of underdetermined set of linear equations, where the solution takes the following closed form

\[ x^{k+1} = (I - KT(KKT)\!^{-1}K)(z^k - u^k) + K^T(KKT)\!^{-1}b \]

(10)

It should be noted that the above solution consists of two main parts; the first term \( x_H = (I - KT(KKT)^{-1})K(z^k - u^k) \) lies in the kernel of matrix \( K \); the second term \( x_P = K^T(KKT)^{-1}b \) is the least norm-2 solution to \( Kx = b \).

It is easy to verify that (10) can be rewritten in the following simplified format

\[ x^{k+1} = u^k + K^T L^{-1} v^k \]

(11)

where \( u^k = z^k - u^k \), \( v^k = b - K u^k \), and \( v^k = L^{-1} v^k \) are new auxiliary variables and \( L = KKT \) is the Laplacian matrix of network graph \( G \). Given \( v^k \), it is straightforward to show that the z-update can be implemented in a distributed way. The k-th iteration of (11) can be interpreted in the following way: the flow \( x^k \) corresponding to the link \( (i, j) \in E \), is updated by adding virtual flow \( w^k \) to the difference value \( sgn(j-i)(v^k_j - v^k_i) \) of nodes at the both end, where \( sgn(a) = 1 \) for \( a \geq 0 \), and \( sgn(a) = -1 \), otherwise. By this interpretation, we only need to develop a distributed method to compute \( v^k \). Hence, the projection step at k-th ADMM iteration is reduced to solving system of linear equations expressed as

\[ r^k = Lu^k \]

(12)

where the direct computation of \( v^k \) requires inversion of Laplacian matrix. We know that the distributed implementation would require each node to have access to the corresponding row. Unlike the Laplacian matrix, its inverse \( L^{-1} \) has nonzero coefficients corresponding to non-neighbor nodes at each row. Hence, the direct inversion solution for (12) requires centralized calculation and therefore global communication of nodes together. Several centralized graph-based algorithms have been proposed for solving symmetric linear equations [9], [10], [11].

Distributed implementation of projection step (11) requires an inner-loop iterative scheme in order to solve linear subproblems (12). A classical iterative scheme which can solve linear system of equations in a distributed fashion is the well-known Jacobi method [12]. However, Jacobi method suffers from slow convergence rate which makes it practically inefficient. We propose a fast and computationally efficient iterative scheme which can converge to desirable accuracy by far fewer iteration steps. Fast convergent iterative scheme proposed in our previous work [7] can be a suitable candidate. This iterative scheme originally inspired by wave propagation in physical media shows superior convergence performance compared to Jacobi.
method. Analysis results show that this scheme significantly improves the convergence rate by reducing the number of iterations from $O(N^2)$ to $O(N)$ for the path network of $N$ nodes. Our proposed algorithm use a second order linear iterative scheme in the following matrix form

$$v^{(l+1)} = (\alpha + 1)Sv^{(l)} - \alpha v^{(l-1)} + c$$ (13)

where $l$ is the inner-loop iteration index, $S = I - D^{-1}L$ is the iteration matrix containing averaging weights, and $c = (1 - \alpha)D^{-1}r^k$ is the bias vector, and $D$ is the degree matrix. The iteration parameter $\alpha$ plays a key role in acceleration of convergence rate. The optimum iteration parameter can be computed for the general network graph (see Appendix A) in the following form

$$\alpha_{opt} = \frac{1 - \sqrt{1 - \lambda_1^2}}{1 + \sqrt{1 - \lambda_1^2}}$$ (14)

where $\lambda_1$ is the spectral radius of matrix $S$. Every node in network need to compute the optimum parameter $\alpha_{opt}$ to perform the accelerated iterations in (13). Hence, all nodes should have the information about spectral radius $\lambda_1$, which depends on the network topology. This fact is in conflict with the aim of distributed implementation where each node just has a degree-1 access to the network structure. In the next section, we resolve this issue by proposing an initialization stage where each node can individually estimate the spectral radius $\lambda_1$ within desired accuracy.

Algorithm 1 gives a concise and elegant presentation of our proposed algorithm as executed at each node. Assuming zero initial values $x_0 = z_0 = v_0 = 0$, after sufficient number of inner-loop iterations $L$ at each node (line 4), the linear subproblem, $Lv^k = r^k$ is solved in a distributed manner. Then, each node updates the set of virtual flows $\{(x_m, z_m, u_m)\}|m \in F_i\}$ connected to the corresponding node in a parallel way (lines 7 to 9), where $F_i = \{m | K_{im} \neq 0\}$. This iterative procedure is repeated until reaching some stopping criterion. Finally, the resulting flow vector $\hat{x}$ is the desired balanced network flow. It should be noted that the optimal parameter is assumed to be already estimated at each node $\hat{\alpha}_i \equiv \alpha_{opt}$ in algorithm 1. The distribution estimation of $\alpha_{opt}$ will be presented in algorithm 2.

### C. Initialization Stage

The global information about network spectral radius required at each node seems to be in conflict with the distributed nature of our algorithm. We design the initialization stage to overcome this obstacle in the path of achieving a fully distributed algorithm. We claim that after sufficient number of iterations, each node can individually estimate $\lambda_1$ almost accurately. It is noteworthy to mention that the proposed initialization routine should be implementable in a distributed way by itself.

The initialization stage starts with the process of averaging iterations, where each node updates its value by averaging the neighbors’ values. These iterations can be easily formulated in the matrix form as follows

$$\bar{x}(n + 1) = S\bar{x}(n) + \bar{u}(n)$$ (15)

where $\bar{x}(n)$ and $\bar{u}(n)$ are vectors of size $N$ containing states of all nodes and the input vector at time $n$ respectively, and matrix $S$ contains averaging weights.

From a purely different viewpoint, an $N$-node network can be considered as a MIMO system of size $N \times N$ with input vector $\bar{u}(n)$ excited at the location of all nodes and the state vector $\bar{x}(n)$ containing observed values at each node at time $n$. The averaging iterations in (15) can be equivalently regarded as the first order linear state-space model for the network system. The basic idea behind the initialization stage is that the dominant mode of system can be extracted from the impulse response. In the case of linear time-invariant system, the solution can be simplified in the following form

$$\bar{x}(n + 1) = S^{n+1}\bar{x}(0) + \sum_{l=0}^{n} S^{n-l}\bar{u}(l)$$ (16)

Assuming zero initial state at all nodes, the impulse response can be written in the following form

$$\bar{x}(n + 1) = S^n\bar{u}(0)$$ (17)

where all nodes are excited by impulse-shaped signals, $\bar{u}_i(n) = \delta(n)$ for $i = 1, \ldots, N$. Since the averaging matrix $S$ is diagonalizable, it can be factorized by eigenvalue decomposition to $S = Q\Lambda Q^{-1}$ where $Q$ and $Q^{-1}$ are square matrices formed of right and left eigenvectors as their columns and rows, respectively, and $\Lambda = \text{diag}[\lambda_1, \ldots, \lambda_N]$ is the diagonal matrix with eigenvalues on the diagonal. Let $q_i$ be the $i$-th column of $Q$ and $\tilde{q}_i$ be the $i$-th row of $Q^{-1}$. The state vector can be expressed as linear combination of various modes of
the system as follows
\[ \tilde{x}(n+1) = QA^nQ^{-1}\tilde{u}(0) = \sum_{i=1}^{N}(\lambda_i)^n\langle \tilde{q}_i|\tilde{u}(0)\rangle q_i \] (18)

where \( \langle ., . \rangle \) is the inner product operation. Suppose the eigenvalues to be indexed in order \( |\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_N| \). Since we have \( \frac{1}{\lambda_i^n} \ll 1, i \in \{2, \ldots, N\} \) for sufficiently large \( n \), the output response will be dominated by the term associated with the largest eigenvalue. Hence, after sufficient number of iterations, the decay rate of output response is determined by the dominant mode
\[ \tilde{x}(n+1) \approx (\lambda_1)^n\alpha_1 q_1 \] (19)

where \( \alpha_1 = \langle \tilde{q}_1|\tilde{u}(0)\rangle \). The expression in (19) verifies the fact that the dominant mode can be extracted from output response of every node in network. By further processing, each node can easily evaluate the dominant mode as follows
\[ \hat{\lambda}_{1,i}^{(n)} = \frac{\tilde{x}_i(n+1)}{\tilde{x}_i(n)} \approx \lambda_1 \] (20)

where \( \hat{\lambda}_{1,i}^{(n)} \) is the estimated dominant mode value from the \( n \)-th iterate of node \( i \). As \( n \) grows large, the dominant role of \( \lambda_1 \) increases which will result in more accurate estimates at all nodes. Furthermore, the accuracy of \( \hat{\lambda}_{1,i}^{(n)} \) estimate can be increased by computing average of dominant modes extracted from a few recent output samples. In our proposed method, each node simply performs \( s \) iterations, then the dominant mode is estimated based on average of last \( r \) estimated values
\[ \hat{\lambda}_1 = \frac{1}{r}\sum_{k=s-r+1}^{s}\hat{\lambda}_{1,i}^{(k)} \] (21)

where \( \hat{\lambda}_1 \) is the vector of size \( N \) containing estimated dominant mode values at all nodes. It should be noted that the process of evaluating dominant mode at each node does not require any further communication between nodes. In other words, every node in network can individually compute an almost accurate estimate of the dominant mode of system or namely spectral radius of matrix \( S \) after sufficient number of iterations. Hence, we achieve an efficient distributed method to estimate the spectral radius of matrix \( S \) and consequently optimum parameter \( \alpha_{opt} \).

Algorithm 2 shows the initialization setup required for the accelerated iterations solving the projection subproblems. As we see, each node computes an estimate of optimum parameter, \( \hat{\alpha}_i \equiv \alpha_{opt} \), after \( s \) averaging iterations.

IV. NUMERICAL RESULTS

We applied the proposed ADMM-based algorithm on a complex network configuration to evaluate the convergence performance. Figure 1 shows a relatively complex graph with set of \( N = 100 \) nodes and \( M = 386 \) edges randomly generated by simulations. Such random generation is inspired by the sensor network nodes, where a node communicates with neighbor nodes not farther than a specific coverage radius. The two farthest nodes are assumed as the source (node 36) and the sink (node 90) nodes. The link capacity is assumed to be the same and normalized to one for all links. A unit rate \( b = 1 \) flow is injected at the source node. It can be seen that the generated network graph has bottleneck with 3 mainstream flows. Figure 2 shows the convergence performance of the proposed ADMM-based method for the generated network graph. To highlight the significance of flows on bottleneck, this figure contains the convergence performance of ten most large-valued flows in network. As it can be seen, our proposed method requires only 3 ADMM iterations to balance network flows on bottleneck within 2% accuracy to the optimal solution. Simulation results show that further ADMM iterations will balance network flows on larger cutsets.

For the initialization stage, we chose the parameters \( s = 60 \) and \( r = 10 \), where every node perform 60 averaging iterations and the dominant mode is estimated based on average of last 10 estimated values. The impulse response of the random generated network system is shown in figure 3 for all nodes. Simulation results show that dominant mode is equal to \( \lambda_1 = 0.9984 \) and the estimated \( \lambda_1 \) achieved from the initialization stage has the mean squared error equal to
\[ MSE = E[(\hat{\lambda}_1 - \lambda_1)^2] = 1.76 \times 10^{-5} \] Hence, all nodes can evaluate the dominant mode within desired accuracy. Network graphs with the dominant mode ratio \( \lambda_2/\lambda_1 \) close to one such as the one we generated, require sufficiently large number of iterations for initialization stage. It should be noted that these iterations are performed only once and would not increase the total computational cost. Finally, it should be mentioned that the value of penalty factor was assumed to be equal to \( \rho = 1 \) for ADMM algorithm.

V. CONCLUSION

We proposed a distributed load balancing protocol based on ADMM iterations for general network configuration. This protocol provides a fully distributed framework where the three-step iterative scheme converges to the minimum \( \ell-p \) solution of network flow. Minimizing the \( p \)-norm, we would yield balanced load distribution throughout the network. We designed an initialization stage where a desired network parameter is estimated in distributed fashion at the location of each node. Our algorithm achieves excellent convergence.
Fig. 1: A typical network graph with $N = 100$ nodes and $M = 386$ edges randomly generated with simulations

Fig. 2: The Convergence performance of proposed ADMM algorithm applied to the generated network in figure 1 for the most significant network flows performance in particular for balancing the traffic load on bottleneck links.

APPENDIX A
OPTIMUM PARAMETER DERIVATION

In order to achieve the best convergence performance, the iteration parameter $\alpha$ should be chosen as (14). Let $v = [v_1, v_2, \ldots, v_N]$ be the vector containing nodes values. The second order iterative scheme (13) can be reduced to first order matrix form as follows,

$$\hat{v}(n+1) = M\hat{v}(n)$$

where $\hat{v}(n+1)$ is a column vector of $2N$ elements and $M$ is matrix of size $2N \times 2N$ defined as,

$$M = \begin{pmatrix} (\alpha + 1)S & -\alpha I_N \\ I_N & 0 \end{pmatrix}, \quad \hat{v}(n+1) = \begin{pmatrix} v(n+1) \\ v(n) \end{pmatrix}$$

Let $\tilde{M}$ be defined as,

$$\tilde{M} = \begin{pmatrix} (\alpha + 1)\Lambda & -\alpha I_N \\ I_N & -\alpha I_N \end{pmatrix}$$

where $\tilde{M}$ is similar to $M$. Therefore, it is required that $|\mu_1| \leq 1$, where $\mu_1$ is the spectral radius of $M$, which has a key role in both stability and convergence rate of iterative scheme. In order to achieve a fast convergent scheme, the absolute value $|\mu_1|$ should be minimized, while satisfying the stability conditions. The spectral radius $|\mu_1|$ relies on $\alpha$ and the spectrum of $S$. Assume $S$ is diagonalizable matrix and can be factorized by eigenvalue decomposition to $S = QAQ^{-1}$, where $Q$ is square matrix with eigenvectors as columns and $\Lambda$ is the diagonal matrix with real eigenvalues on the diagonal, $\Lambda_{ii} = \lambda_i$. By this decomposition, matrix $M$ can be factorized to,

$$M = \begin{pmatrix} Q & 0 \\ 0 & Q \end{pmatrix} \begin{pmatrix} (\alpha + 1)\Lambda & -\alpha I_N \\ I_N & 0 \end{pmatrix} \begin{pmatrix} Q^{-1} & 0 \\ 0 & Q^{-1} \end{pmatrix}$$

Let $\hat{M}$ be defined as,

$$\hat{M} = \begin{pmatrix} (\alpha + 1)\Lambda & -\alpha I_N \\ I_N & 0 \end{pmatrix}$$

Since $M \sim \hat{M}$ are similar matrices, they share the same set of eigenvalues. Moreover, $\hat{M}$ can be decomposed to $2 \times 2$ matrices for each eigenvalue $\lambda_i$ as below,

$$\begin{pmatrix} (\alpha + 1)\lambda_i & -\alpha \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \mu_i \begin{pmatrix} x \\ y \end{pmatrix}$$

Fig. 3: Impulse response of generated network graph at the location of all nodes
where each eigenvalue $\mu_i$ can be simply found as root of the characteristic polynomial,

$$\mu_i^2 - (\alpha + 1)\lambda_i \mu_i + \alpha = 0 \quad (27)$$

Note that each $\lambda_i$ is mapped to a pair of eigenvalues $\mu_i^\pm = \gamma_i \pm \sqrt{\gamma_i^2 - \alpha}$, where $\gamma_i = (\alpha + 1)\lambda_i/2$. To achieve optimum convergence rate, the parameter $\alpha$ should be chosen such that $|\mu_1|$ is minimized. The spectral radius of $S$ is denoted as $\lambda_1$.

For the case when $\gamma_1^2 > \alpha$, there exists $|\mu_1| > |\gamma_1|$. In the other case, when $\gamma_1^2 \leq \alpha$, it will lead to complex conjugate pair, $\mu_i^\pm = \gamma_i \pm j\sqrt{\alpha - \gamma_i^2}$ where $|\mu_i^\pm| = \sqrt{\alpha}$. Hence, the lower bound for $|\mu_1|$ is achieved by choosing $\alpha$ to be equal to

$$\alpha_{\text{opt}} = \gamma_1^2 \quad (28)$$

Substituting $\gamma_1 = (\alpha + 1)\lambda_1/2$, we will have,

$$\lambda_1^2 = \frac{4\alpha_{\text{opt}}}{(\alpha_{\text{opt}} + 1)^2} \quad (29)$$

If we define the stability factor as $\xi = \frac{1 - \alpha}{1 + \alpha}$, then (29) can be rewritten as,

$$\xi_{\text{opt}}^2 = 1 - \lambda_1^2 \quad (30)$$

Hence, the optimum parameter $\alpha_{\text{opt}}$ which attains optimal convergence performance is equal to,

$$\alpha_{\text{opt}} = \frac{1 - \xi_{\text{opt}}}{1 + \xi_{\text{opt}}} = 1 - \frac{\sqrt{1 - \lambda_1^2}}{\sqrt{1 + \lambda_1^2}} \quad (31)$$

REFERENCES


